Contents

0. Preliminaries.

Main environment variables

We use environment variables to make these instructions as location-invariant as possible.
MOAB is part of a larger development effort centered at Argonne and called Fathom,
therefore, for convenience, we will install MOAB all of the packages required by MOAB under
${FATHOM_DIR}.

This organization is, of course, optional. As a new package *PACKAGE* satisfying
a MOAB dependency is installed, its full installation directory path is stored in
*PACKAGE*_PREFIX environment variable. The building and installation of further packages
depends on *PACKAGE* only through ${*PACKAGE*_PREFIX} and can proceed independently
of how *PACKAGE* was built.

Compilers

In these instructions we illustrate the building and installation procedure using
the *GNU Compiler Collection (gcc)*. The collection includes *gcc , g++, gfortran*
(note that *gcc* is the name of the collection *and* of the GNU C compiler). An essentially
identical procedure should work on most Linux machines (e.g., a Linux laptop) and with
different compilers (e.g., Intel's compilers).

We assume from the outset that we wish to build a *parallel* version of MOAB.
A typical (although not minimal) set of prerequisite packages, which supports parallel MOAB
capable of reading and writing the most common file formats is: MPI2, ZLIB, SZIP, HDF5, NETCDF.
In particular, MPI2 (an implementation of the MPI-2 standard) must underly all of the libaries with
parallel capabilities, including HDF5 and NETCDF. In particular, since HDF5 requires MPI2
capabilities (MPI-IO), MPI1 will not suffice.

It is important to mention that all of the dependencies must be build with a consistent set of compilers.
If different packages are built with different compilers (e.g., MPI2 with gcc and HDF5 with Intel's compilers),
linking problems may occur. In particular, different packages will end up using different implementations
of *libc* -- the C Standard Library.

Similarly, all of the packages using MPI2 should be linked against the same MPI2 library. In fact, a convenient
way to ensure compiler consistency is to use MPI compilers (*mpicc, mpicxx, mpif77, mpif90*) to build *all* of
MOAB-related packages. There is no harm in building serial libraries (e.g., ZLIB) or applications using MPI
compilers. In fact, MPI compilers simply wrap the compilers used to build MPI (e.g., gcc) and add extra directives,
ensuring the MPI-related declarations and symbols are correctly resolved and link. If no MPI calls are present,
MPI compilers behave just like the underlying basic compilers and generate no extraneous code.
For these reasons, we build MPI2 first, and all of the other packages are then built using the MPI compilers.

This way both compiler *and* MPI consistency is enforced.

We separate package installations based on different MPI2 builds by using the ${BUILD} label, which contains a unique identifier of the underlying MPI2 build. Once MPI2 has been built, the MPI2_PREFIX and BUILD environment
variable are defined and the rest of the packages are built.

Thus, FATHOM_DIR, MPI2_PREFIX, BUILD and all of the *PACKAGE*_PREFIX variables are the only environment variables that have to be carried over from the installation of one package to another. Other environment variables used in these instructions are there convenience only and to help maintain and modify the script as various parameters, such as package versions, change.

Directory structure

For the sake of the presentation we assume the package organization structure described here.

- Each dependency *PACKAGE* is downloaded, built and installed under ${FATHOM_DIR}/*package*.
    - The source code goes under ${FATHOM_DIR}/*package*/*package_version*, where *package_version*

is typically (but not always) of the form *package-version.major.minor*.

- The package itself is installed into ${FATHOM_DIR}/*package*/*package_version*/'${BUILD},

where *configuration* will typically reflect the choice of compilers used to build the package.

For example, if we are using the mpich2-1.2.1 implementation of MPI2, we can designate the build by BUILD=mpich2-1.2.1/gcc. Then, when the package being built *PACKAGE* is HDF5 and we are using its version hdf5-1.8.3, we will have the source in ${FATHOM_DIR}/hdf5/hdf5-1.8.3 and will install the package in
HDF5_PREFIX=${FATHOM_DIR}/hdf5/hdf5-1.8.3/mpich2-1.2.1/gcc.

In order to facilitate the maintenance of these instructions and the resulting installation scripts, we will use an auxiliary environment variables *PACKAGE*_VERSION and, occasionally, *PACKAGE*_VERSION_NUMBER to store *package_version* and *major.minor*.

Autotools

All of the packages described below are built using the standard autotools based using the typical *configure; make; make install* procedure with the corresponding *configure*, *make*, and *install* stages. Occasionally, extra stages, such as *make check* or *make test* may intervene, and will be specifically mentioned.

Logs

We ensure that there are logs of all of the build stages -- *configure.log, make.log, make.install.log*, and others such as *make.check.log*, if applicable. After the build and installation are completed, all the relevant log files are copied into ${*PACKAGE*_PREFIX}.

1. Building MPI2

We use the *mpich2* -- the free implementation of MPI-2 available from
Argonne National Laboratory: http://www.mcs.anl.gov/mpich.
If a different MPI2 implementation is desired (e.g., vendor-provided), set MPI2_PREFIX accordingly
and skip this section.

The `1.2.1` version of mpich2 can be downloaded, built using *gcc* and installed in accordance with
the above-discussed naming conventions using the following command sequence:

```
mkdir --parents ${FATHOM_DIR}/mpi2

wget -O ${FATHOM_DIR}/mpi2/mpich2-1.2.1.tar.gz http://www.mcs.anl.gov/research/projects/mpich2/downloac

tar zxv --file ${FATHOM_DIR}/mpi2/mpich2-1.2.1.tar.gz --directory ${FATHOM_DIR}/mpi2

rm -f ${FATHOM_DIR}/mpi2/mpich2-1.2.1.tar.gz

cd ${FATHOM_DIR}/mpi2/mpich2-1.2.1

export CC=gcc
export CXX=g++
export F77=gfortran
export FC=gfortran
export F90=gfortran

export MPI2_PREFIX=${FATHOM_DIR}/mpi2/mpich2-1.2.1/gcc

./configure --prefix=${MPI2_PREFIX} 2>&1 | tee configure.log

make 2>&1 | tee make.log

make install 2>&1 | tee make.install.log

make installcheck 2>&1 | tee make.installcheck.log

cp configure.log make.log make.install.log make.installcheck.log ${MPI2_PREFIX}
```

Caveats: There a few things that might have to be done for MPI2 jobs to run correctly, in particular, to make sure *make installcheck* works.

- MPI2 (at least *mpich2*), unlike MPI1 (and *mpich1*) uses *mpd* demons to start parallel jobs.
- *mpd* might not start unless ${HOME}/.mpd.conf exists and contains an MPD_SECRETWORD entry.

For more information see the mpich2 documentation here.
In the meantime, however, the following procedure should make sure that you can run mpi jobs, at least *make installcheck*
should succeed:
1. Create ${HOME}/.mpd.conf and make sure it is not readable by anybody except you:

```
touch ${HOME}/.mpd.conf
chmod og-rwx ${HOME}/.mpd.conf
```

Then add the following line to ${HOME}/.mpd.conf:

```
MPD_SECRETWORD=ty7-q9z
```

where any word of a similar form can be used instead of 'ty7-q9z' (see documentation for more details).
2. Start an *mpd* demon before you run parallel jobs or *make installcheck*. It may be a good idea to stop the demon afterwards.
With this in mind, the above `make installcheck` should be replaced by something like:

```
${MPI2_PREFIX}/bin/mpdboot -n 1                          # this will start a single mpd on your machin
make installcheck 2.&1 | tee make.installcheck.log  # or other parallel runs, such as ${MPI2_PREFIX}/bi
${MPI2_PREFIX}/bin/mpdallexit                             # this will cause mpd to exit
```

It may be a good idea to add ${MPI2_PREFIX}/bin to your PATH in your `${HOME}/.bash_profile` or `${HOME}/.bashrc`
(or the equivalent resource file for the shell of your choice):

```
export PATH=${MPI2_PREFIX}/bin:${PATH}
```

This will simplify the running of parallel jobs later. For the purposes of building MOAB, however, ${MPI2_PREFIX} is
needed as well, so we do not make assumptions about your PATH and use ${MPI2_PREFIX} explicitly throughout.

2. Building other MOAB prerequisites

Now that MPI2 has been built, we can instruct the remaining packages to use the MPI2 compilers and define the build variable to label the package installations corresponding to our version of MPI2:

export BUILD=mpich2-1.2.1/gcc export CC=${MPI2_PREFIX}/bin/mpicc export CXX=${MPI2_PREFIX}/bin/mpicxx export F77=${MPI2_PREFIX}/bin/mpif77 export FC=${MPI2_PREFIX}/bin/mpif90 export F90=${MPI2_PREFIX}/bin/mpif90

We will redundantly replicate the compiler definitions below, making sure that any section of this page can be used independently to build the corresponding package, provided ${FATHOM_DIR}, ${MPI2_PREFIX} and ${BUILD} are define (in addition to the prefixes of the packages that the package being built depends on).

Building ZLIB

We use *zlib*, the canonical implementation of ZLIB available from http://www.zlib.net.
If a different ZLIB implementation is desired (e.g., already installed)
set ZLIB_PREFIX accordingly and skip this section.
Remember, however, the rant about compiler compatibility at the beginning of this section.

The `1.2.3` version of zlib can be downloaded and installed in accordance with the above-discussed naming conventions using the following command sequence:

```
mkdir --parents ${FATHOM_DIR}/zlib

export ZLIB_VERSION=zlib-1.2.3

wget -O ${FATHOM_DIR}/zlib/${ZLIB_VERSION}.tar.gz  http://www.zlib.net/${ZLIB_VERSION}.tar.gz
```

```
tar zxv --file ${FATHOM_DIR}/zlib/${ZLIB_VERSION}.tar.gz --directory ${FATHOM_DIR}/zlib

rm -f ${FATHOM_DIR}/zlib/${ZLIB_VERSION}.tar.gz

cd ${FATHOM_DIR}/zlib/${ZLIB_VERSION}


export CC=${MPI2_PREFIX}/bin/mpicc
export CXX=${MPI2_PREFIX}/bin/mpicxx
export F77=${MPI2_PREFIX}/bin/mpif77
export FC=${MPI2_PREFIX}/bin/mpif90
export F90=${MPI2_PREFIX}/bin/mpif90
export ZLIB_PREFIX=${FATHOM_DIR}/zlib/${ZLIB_VERSION}/${BUILD}

./configure --prefix=${ZLIB_PREFIX} 2>&1 | tee configure.log

make 2>&1 | tee make.log

make install 2>&1 | tee make.install.log

cp configure.log make.log make.install.log ${ZLIB_PREFIX}
```

Building SZIP

We use *szip*, the canonical implementation of SZIP available from http://www.compressconsult.com.[[BR]] If a different SZIP implementation is desired (e.g., already installed)
set SZIP_PREFIX accordingly and skip this section.
Remember, however, the rant about compiler compatibility at the beginning of this section.

Legal disclaimer: SZIP source can be used for **non-commercial** purposes only[[BR]] See more about this here.
You are responsible for making sure you can use the SZIP source legally.

The `2.1` version of szip can be downloaded and installed in accordance with the above-discussed naming conventions using the following command sequence:

```
mkdir --parents ${FATHOM_DIR}/szip

export SZIP_VERSION_NUMBER=2.1
export SZIP_VERSION=szip-${SZIP_VERSION_NUMBER}

wget -O ${FATHOM_DIR}/szip/${SZIP_VERSION}.tar.gz ftp://ftp.hdfgroup.org/lib-external/szip/${SZIP_VERSI

tar zxv --file ${FATHOM_DIR}/szip/${SZIP_VERSION}.tar.gz --directory ${FATHOM_DIR}/szip

rm -f ${FATHOM_DIR}/szip/${SZIP_VERSION}.tar.gz

cd ${FATHOM_DIR}/szip/${SZIP_VERSION}

export SZIP_PREFIX=${FATHOM_DIR}/szip/${SZIP_VERSION}/gcc

export CC=${MPI2_PREFIX}/bin/mpicc
export CXX=${MPI2_PREFIX}/bin/mpicxx
export F77=${MPI2_PREFIX}/bin/mpif77
export FC=${MPI2_PREFIX}/bin/mpif90
export F90=${MPI2_PREFIX}/bin/mpif90


./configure --prefix=${SZIP_PREFIX} 2>&1 | tee configure.log
```

```
make 2>&1 | tee make.log

make check 2>&1 | tee make.check.log

make install 2>&1 | tee make.install.log

cp configure.log make.log make.check.log make.install.log ${SZIP_PREFIX}
```

Building HDF5

We use *hdf5*, the canonical implementation of HDF5 available from http://www.hdfgroup.org/.
If a different HDF5 implementation is desired (e.g., already installed)
set HDF5_PREFIX accordingly and skip this section.
Remember, however, the rant about compiler compatibility at the beginning of this section.
HDF5 depends on ZLIB and SZIP.

The `1.8.3` version of hdf5 can be downloaded and installed in accordance with the above-discussed naming conventions using the following command sequence:

```
mkdir --parents ${FATHOM_DIR}/hdf5

export HDF5_VERSION=hdf5-1.8.3

wget -O ${FATHOM_DIR}/hdf5/${HDF5_VERSION}.tar.gz  http://www.hdfgroup.org/ftp/HDF5/prev-releases/${HDF

tar zxv --file ${FATHOM_DIR}/hdf5/${HDF5_VERSION}.tar.gz --directory ${FATHOM_DIR}/hdf5

rm -f ${FATHOM_DIR}/hdf5/${HDF5_VERSION}.tar.gz

cd ${FATHOM_DIR}/hdf5/${HDF5_VERSION}

export HDF5_PREFIX=${FATHOM_DIR}/hdf5/${HDF5_VERSION}/gcc

export CC=${MPI2_PREFIX}/bin/mpicc

./configure --enable-parallel --prefix=${HDF5_PREFIX} --with-zlib=${ZLIB_PREFIX} --with-szlib=${SZIP_PR

make 2>&1 | tee make.log

make check 2>&1 | tee make.check.log

make install 2>&1 | tee make.install.log

cp configure.log make.log make.check.log make.install.log ${HDF5_PREFIX}
```

Building NETCDF

We use *netcdf*, the canonical implementation of netCDF available from
http://www.unidata.ucar.edu/software/netcdf/.
If a different netCDF implementation is desired (e.g., already installed)
set NETCDF_PREFIX accordingly and skip this section.
Remember, however, the rant about compiler compatibility at the beginning of this section.
netCDF depends on HDF5, ZLIB and SZIP.

The `4.1` version of netcdf can be downloaded and installed in accordance with the above-discussed naming conventions using the following command sequence:

```
mkdir --parents ${FATHOM_DIR}/netcdf

export NETCDF_VERSION=netcdf-4.1

wget  -O ${FATHOM_DIR}/netcdf/${NETCDF_VERSION}.tar.gz http://www.unidata.ucar.edu/downloads/netcdf/ftp

tar zxv --file ${FATHOM_DIR}/netcdf/${NETCDF_VERSION}.tar.gz --directory ${FATHOM_DIR}/netcdf

rm -f ${FATHOM_DIR}/netcdf/${NETCDF_VERSION}.tar.gz

cd ${FATHOM_DIR}/netcdf/${NETCDF_VERSION}

export NETCDF_PREFIX=${FATHOM_DIR}/netcdf/${NETCDF_VERSION}/gcc

export CC=${MPI2_PREFIX}/bin/mpicc
export CXX=${MPI2_PREFIX}/bin/mpicxx

./configure --prefix=${NETCDF_PREFIX} --disable-shared --disable-f77 --disable-f90 --enable-netcdf-4 --

make 2>&1 | tee make.log

make check 2>&1 | tee make.check.log

make install 2>&1 | tee make.install.log

cp configure.log make.log make.check.log make.install.log ${NETCDF_PREFIX}
```

Caveats: There are several things that have to be done to ensure that *netcdf* builds correctly *and* *moab* correctly links against *netcdf*

- In the past we have experienced problems with compiling Fortran sources for *netcdf*, so we chose to disable them.
- *hdf5* appears to be only available as static archives (.a), not as shared libraries (.so), so we disabled shared in *netcdf* as well.

Disabling Fortran should be safe, since MOAB is implemented in C++. Even if Fortran bindings for MOAB appear at a later time, internally only C/C++ bindings to *netcdf* will be used, so no Fortran is required of *netcdf*.

In the past we found that if *hdf5* is linked statically (there appears to be no shared build version of *hdf5*), and *netcdf* libraries use shared linking, configure tools (*libtool*, to be precise) will simply omit one of or both *hdf5* and *netcdf*. That might be a *libtool* bug that needs fixing, but until it is fixed, the above is an acceptable "workaround" (in our opinion).

2. Building MOAB trunk

Obtaining MOAB source

The MOAB source can be obtained using Subversion version control system, also known as *svn*. Subversion will allow you to check out MOAB's source straight from the main development repository: https://svn.mcs.anl.gov/repos/ITAPS/MOAB.

There are several versions or *branches* of MOAB in the main repository: the main branch, the *trunk*, and several other branches, including stable releases and various specialized versions.

Use the following command to check out the *trunk* branch using svn and copy it to the local directory ${FATHOM_DIR}/moab/trunk

```
mkdir --parents ${FATHOM_DIR}/moab
svn co https://svn.mcs.anl.gov/repos/ITAPS/MOAB/trunk ${FATHOM_DIR}/moab/trunk
```

The above fits with our established terminology as follows: if MOAB is considered a *PACKAGE*, then *trunk* is its *version* correctly downloaded under ${FATHOM_DIR}/moab.

Building MOAB

MOAB is a *research* code, so it is natural that, unlike its prerequisites, MOAB is not distributed as a ready-to-configure GNU package. Therefore, the steps necessary to configure and build it are slightly different. The difference is not substantial, however: before MOAB can be configured, we have to use autotools to generate the necessary *configure* and *makefile* files.
This is accomplished by calling '`autoreconf -fi` in ${FATHOM_DIR}/moab/trunk (or from another branch's source directory). A full sequence of commands sufficient to configure, build and install *trunk* with the above prerequisites is as follows:

```
cd ${FATHOM_DIR}/moab/trunk

autoreconf -fi

export MOAB_PREFIX=${FATHOM_DIR}/moab/trunk/gcc

export CC=${MPI2_PREFIX}/bin/mpicc
export CXX=${MPI2_PREFIX}/bin/mpicxx

./configure --prefix=${FATHOM_PREFIX} --with-mpi=${MPI2_PREFIX} --with-netcdf=${NETCDF_PREFIX} --with-h

make 2>&1 | tee make.log

make check 2>&1 | tee make.check.log

make install 2>&1 | tee make.install.log

cp configure.log make.log make.check.log make.install.log ${MOAB_PREFIX}
```

Remarks

Discuss moab.conf.sh and the build scripts here.